# NAVAL POSTGRADUATE SCHOOL

## Monterey, California


STRUCTURE AND ERROR DETECTION IN COMPUTER SOFTWARE

by

Gordon H. Bradley, Thomas Green, Gilbert T. Howard
and Norman F. Schneidewind

February 1975

Approved for public release; distribution unlimited.

Prepared for:
Naval Air Development Center
Warminster, Pennsylvania

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Isham Linder
Superintendent

Jack R. Borsting
Provost

The work reported herein was supported by the Naval Air Development Center, Warminster, Pennsylvania.



This report was prepared by:

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Date Entered)*

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER: NPS55Ss75021 | 2. GOVT ACCESSION NO. |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE *(and Subtitle)*: Structure and Error Detection in Computer Software | 5. TYPE OF REPORT & PERIOD COVERED: Technical Report |
| | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR*(s)*: Gordon H. Bradley, Thomas Green, Gilbert T. Howard, Norman F. Schneidewind | 8. CONTRACT OR GRANT NUMBER*(s)* |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: N62269/75/RO/02014 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Air Development Center, Warminster, Pennsylvania | 12. REPORT DATE: February 1975 |
| | 13. NUMBER OF PAGES |
| 14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* | 15. SECURITY CLASS. *(of this report)* |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A model of the error detection process for the testing of software has been developed to investigate the relationship of computer program structure to error detection and test effort. The model has been implemented as a simulation. Analytical results have also been obtained.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

# STRUCTURE AND ERROR DETECTION IN COMPUTER SOFTWARE*

Gordon H. Bradley, Thomas Green, Gilbert T. Howard
and Norman F. Schneidewind

## ABSTRACT

A model of the error detection process for the testing of software has been developed to investigate the relationship of computer program structure to error detection and test effort. The model has been implemented as a simulation. Analytical results have also been obtained.

## INTRODUCTION

Software costs are an increasing fraction of total computer project costs and for many computer projects, software costs dominate hardware costs. Much of the software development costs are for testing, debugging and integration and a significant part of the costs after releasing the software are for correcting errors. Thus there is current interest in the error characteristics of software. It is generally accepted that computer programs with complex structure are harder to debug and test and more errors persist after release than for programs with more simple structure. Here we develop a model** to investigate the relationship of program structure to error detection and test effort.

Since structure can be controlled during the design phase and measured through all phases of a computer project, the study of the relationship between structure and error characteristics is valuable to the manager of a software project. Complex program structures with poor error characteristics should

---

* This work was supported by a contract from the Naval Air Development Center, Warminster, Pennsylvania.

** The suggestion to use a simulation model to study software error detection was given to the authors by Dr. Samuel Litwin, a consultant to the Navy Air Development Center.

be avoided. In cases where complex program structures may be necessary to help meet program size or speed limitations, it is useful to have an indication of the additional testing which may be caused by complex structures. It is also useful to be able to compare the error characteristics of design alternatives that have different program structure.

## SOFTWARE COSTS

The cost of computer systems can be divided into hardware (cost of computers and peripheral equipment) and software (cost to design, write, test and maintain computer programs). The proportion of total costs that is attributable to software has been increasing; projections of higher costs for people and lower costs for hardware components indicate that this trend will continue. The direct cost of software is enormous; one estimate of the cost in the United States (1) is 10 billion dollars per year. Indirect costs due to late delivery, nonperformance due to errors, wrong actions due to erroneous output, etc., are also very large.

Although there is not extensive data available on the cost of software projects, there is enough data to indicate the magnitude of software costs. A comprehensive study of Air Force computer costs (1), (2) estimates that software costs in 1972 were from 1 to 1.5 billion dollars, which is 4% to 5% of the total Air Force budget, while computer hardware costs were from 300 to 400 million dollars. The percentage of computer costs due to software has increased from 30% in the late 1950's to 70% now and the estimate for 1985 is 90%. The software costs for the U. S. manned space missions from 1960 to 1970 have been estimated at 1 billion dollars (1). In the private sector the same magnitudes and trends are present. The development cost of a modern general purpose computer is about one half for hardware and one half for software

(operating systems, compilers, support programs). The operating system developed by IBM for the 360 series is estimated to have cost 200 million dollars or about 1000 dollars per instruction in the final version. The indirect costs were also enormous: the project was one year late.

The production of software can be divided into three phases, 1) analysis and design, 2) writing programs and 3) test and integration. Data on how time, effort and money are divided among these three phases gives some indication of why software production is so costly. The fraction of time, effort and money for each phase differs from application to application; however, data from some large projects show similar experience. Estimates are given in (1), (2) for some military command and control systems: analysis and design is about 35%, writing programs 15% and test and integration 50%. For space projects the estimates are 35%, 20%, 45%. For the IBM 360 operating system the estimates are 35%, 15%, 50%. Data for business applications indicates less for testing and integration and more for analysis and design than the above data. The surprising amount of time, effort and money for test and integration is often the item most underestimated in planning computer projects.

## TESTING AND ERROR DETECTION

In many moderate and large computer projects, a programmer writes and debugs a module and then gives it to a test group. The test group tests the module, integrates it with other modules and then continues testing. The module is tested by supplying an input to the module and then comparing the outcome to the known correct outcome. If there is a mismatch between observed and correct output, an error has been detected. When an error is detected the module is given to a programmer who locates and corrects the error (i.e. debugs the module) and then returns the module to the test group. Notice the

distinction between testing, which is supplying inputs and observing outputs and debugging, which is the highly individualized detective work needed to locate and correct errors. In debugging, the programmer needs a detailed knowledge of the structure and operation of the module. The tester is frequently unaware of module structure and operation; he needs only to understand the function of the module.

Most computer programs have a large number of potential inputs; each may exercise the program in a different way. The sequence of instructions of the program that results from a particular input is called the "path" or "thread" associated with that input. Testing by submitting inputs to the program checks only the paths associated with those inputs. For programs with a very large number of inputs, testing can only be a relatively small sampling of all possible inputs.

## ERROR DETECTION MODEL

Testing is a critical part of software projects because it measures and affects the final quality of the software and it consumes a large part of project time and resources. Testing also reveals the strengths and weaknesses of the analysis, design and coding of the software and gives an estimate of the success or failure of the software after release. Thus it is important to understand the testing process and to understand the relationships between testing and the various decision variables that may be controlled during analysis, design and coding. Here we investigate how error detection during testing is affected by the structure of a computer program. By structure we mean how the parts of the program are related.

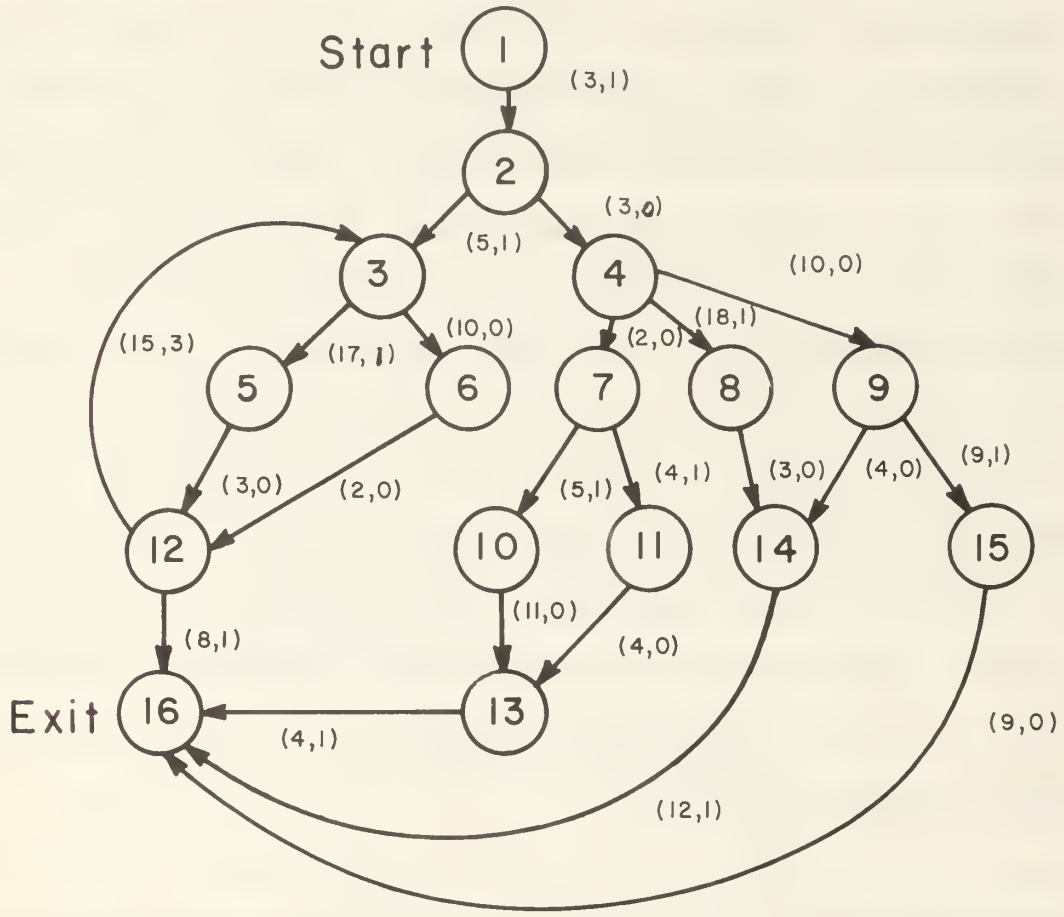
It is very difficult to do experimentation with program structure in actual software projects because the cost of duplicate implementations of the

same application is very high for all but small projects. For this reason analysis is performed on a model.

Structure may be modeled as a set of nodes and arcs as shown in Figure 1. In the directed graph, nodes represent connection points where parts of the program may merge and/or branch and arcs represent a sequence of nonbranching instructions such as computation and input/output. Instructions are located in arcs and errors are located in some of the instructions. An input defines a path from the start node to the exit node. Beginning at the start node an input causes execution of the instructions on its path, consuming test time, until an error is encountered. After the error is thus detected, it is corrected, consuming correction time; there is some risk that the correction will introduce a new error in some instruction. Then restarting at the initial node execution is begun again with the same input. This process is repeated until there are no errors on the path. For each branch node the probability of selection of each arc is known.

A basic assumption of the model is that the tester has some knowledge of the program structure but that for a given input he does not know the specific path that it will execute. In actual software projects the test group has flow charts and program listings. However, it is infeasible to analyze this information because it may contain thousands of lines of coding. Because of the size of the program, the complicated internal logic and the very large number of paths, the relationship between inputs and outcomes is rarely understood. One example is in the testing and maintenance of large operating systems; the relationship of inputs to outcomes is so poorly understood that even after an error has been detected it is often difficult to determine an input that will reproduce the error.

13 Errors originally; Number of instructions is 161.
( , ) indicates number of instructions, followed by number of errors in arc. E Signifies an error found and NE Signifies a new error inserted in designated arc.




Input 1 Traversals: 1→2, E; 1→2→3,E; 1→2→3→6→12→3, E;

NE, 9→14; 1→2→3→6→12→3, E; NE, 9→14; 1→2→3→6→12→3, E;
NE, 9→14; 1→2→3→6→12→3→6→12→16, E; 1→2→3→6→12→

Execution time = 6.59 hrs, Repair time = 395.67 mins.

6 Errors found, 3 Errors inserted, 10 Errors remain.

Input 2 Traversals: 1→2→3→6→12→3→5, E;

1→2→3→6→12→3→5→12→3→6→12→3→5→12→16.

Execution time = .061 hrs, Repair time = 3.65 mins.

1 Error found, 9 Errors remain.

Input 3 Traversals: 1→2→3→5→12→16.

Execution time ≈ 0 hrs, Repair time = 0 mins.

0 Errors found, 9 Errors remain.

Figure 1 -- Simulation Example ( 3 Inputs Used)

A further assumption of the model is that the tester gains no information as the testing proceeds that will influence his choice of subsequent inputs. In actual software projects the tester should try to make best use of any information gained during testing. Various software packages are available for recording the following types of data: count and frequency distributions of instructions executed, indication of code that is not executed and indication of code that is impossible to reach (3). However, there are other factors that may make it difficult to effectively use the information gained during testing; for example, the test plan may be specified in advance with no modifications allowed or inputs may be restricted to those that will be typical for the program in actual operation. For these reasons the model assumptions seem reasonable.

Program structure affects the error detection process; to study this relationship it is helpful to have measures of each. For the error detection process some measures are: number of errors detected in a fixed time, number of errors detected with a fixed number of inputs, mean time between errors, percent arcs traversed by one or more inputs and percent errors remaining. More sophisticated measures involving the shape of the graph of errors detected vs time are also possible. These measures will be called "error detection characteristics." Good measures of program structure are harder to define. The most simple measure is size as measured by the number of nodes. A measure that expresses the degree of completeness of the graph is the ratio of the actual number of arcs in the graph to the maximum possible number with only one arc between each pair of nodes. Since the model allows parallel arcs this number can be greater than one. These measures will be called "complexity measures."

The model can be used to influence the software design decisions on program structure by making it possible to compare the error detection characteristics of design alternatives. Since error detection characteristics are good indicators of the time and resources consumed by testing, it is valuable to relate design decisions made early in the project to testing. The design flow charts and estimates of branch probabilities and number of instructions can be used to specify the model; the model is then seeded with errors and subjected to random inputs.

The model can also be used to identify the measure or measures of complexity that best predict error detection characteristics. To do this, it is necessary to gather data from the model on the error detection characteristics of a variety of different structures and then do a statistical analysis. This would make it possible to measure the complexity of different programs and then compare the estimates of error detection characteristics. Although some data has been generated, further work is necessary to identify good measures of complexity.

There are other situations where it is useful to be able to compare structures. A frequent problem is to evaluate the cost of adding some additional feature to the program. The results of the model can be used to compare the error detection characteristics of the original and modified structure. The problem of how to allocate test effort among projects of different size and different structure can also be addressed.

Since measures of complexity are to be used to estimate error detection characteristics, it is important to define measures that adequately express the differences between structures with good and poor error detection characteristics. Since inputs are associated with paths, a measure of complexity is the

number of paths. The average number of arcs per path also is a measure that is related to the number of errors detected per input. For moderate size graphs with no directed cycles, it is easy to enumerate all the paths and the number of arcs on each. If there are directed cycles, it is necessary to put an upper limit on the number of arcs in the paths considered in order to eliminate paths with an uncountable number of arcs. The number of ways that an arc can be reached indicates how accessible it is to testing. Measures based on this are the mean and standard deviation of the number of paths that traverse each arc or the number of arcs that are traversed by less than a fixed number of paths.

The complexity measures defined thus far depend only on the topology of the directed graph; it is also possible to use the branch probabilities. Since the paths are not equally likely, the measures involving paths can be weighted by the path probabilities. Given the probability that an arc will be traversed by a single input, a complexity measure of the accessibility of the arcs for testing is the sum of these probabilities for all arcs.

## ANALYTICAL RESULTS

In the analysis of the model, we want to study the effect of structure on the number of errors detected by a sequence of inputs. For a graph with the branch probabilities known and the expected number of errors on each arc known, it is possible to calculate the expected number of errors detected by each of a sequence of inputs (recall that an input involves the one or more trys necessary to obtain an error free path from the start node to the exit node). The analytical results are a relatively inexpensive means to analyze the relationship between structure and the error detection process. The analytical results can also help in the statistical analysis of the

simulation discussed below and can reduce the number of simulation runs needed.

To simplify the analysis, it is assumed that new errors are not created by the correction of errors. Figure 2 shows the results of the analysis of the directed graph of Figure 1 where the expected number of errors in each arc is 0.6 and for each branch node the probability of taking each arc is equal.

INITIAL EXPECTED NUMBER OF ERRORS= 13.20

| INPUT | EXPECTED NUMBER OF ERRORS DETECTED | CUMMULATIVE EXPECTED NUMBER OF ERRORS DETECTED |
|---|---|---|
| 1 | 3.45 | 3.45 |
| 2 | 1.95 | 5.40 |
| 3 | 1.39 | 6.79 |
| 4 | 1.04 | 7.82 |
| 5 | 0.80 | 8.62 |
| 6 | 0.63 | 9.25 |
| 7 | 0.51 | 9.76 |
| 8 | 0.42 | 10.19 |
| 9 | 0.36 | 10.54 |
| 10 | 0.30 | 10.85 |
| 11 | 0.26 | 11.10 |
| 12 | 0.22 | 11.33 |
| 13 | 0.20 | 11.52 |
| 14 | 0.17 | 11.69 |
| 15 | 0.15 | 11.84 |
| 16 | 0.13 | 11.98 |
| 17 | 0.12 | 12.10 |
| 18 | 0.11 | 12.20 |
| 19 | 0.09 | 12.30 |
| 20 | 0.08 | 12.38 |

Figure 2 - Expected number of errors detected in Figure 1.

It is necessary to calculate the probability that each arc will be traversed by an input that begins at the start node. This calculation is straightforward for arcs that can be traversed at most once by a single input. The probability of reaching a node is the sum of the traversal probabilities for all arcs coming into the node and the traversal

probability of arcs leaving the node is the branch probability of the arc times the probability of reaching the node. For arcs that can be traversed more than once by a single input (called repeated arcs), the calculation is more complex. In Figure 1, because arc $12 \rightarrow 3$ loops back, arcs $3 \rightarrow 5$, $3 \rightarrow 6$, $5 \rightarrow 12$, $6 \rightarrow 12$ and $12 \rightarrow 3$ are repeated arcs. The probability of reaching node 3 is 1/2, but the probability of traversing arc $3 \rightarrow 5$ is greater than 1/4 because even if an input traverses arc $3 \rightarrow 6$, there is some chance that node 3 will be revisited via $12 \rightarrow 3$ and that arc $3 \rightarrow 5$ could then be traversed. The probability that an input will traverse $3 \rightarrow 6$, $6 \rightarrow 12$, $12 \rightarrow 3$ and then traverse $3 \rightarrow 5$ is $(1/4)(1/2)(1/2) = 1/16$. The probability of traversing arc $3 \rightarrow 6$ twice and then traverse $3 \rightarrow 5$ is $(1/16)(1/2)(1/2) = 1/64$. Thus, the probability of traversing arc $3 \rightarrow 5$ is $1/4 + 1/16 + 1/64 + \ldots = 1/3$.

The calculation of the traversal probabilities can be done with a Markov chain analysis where each arc is a state and there is an additional, fictitious, absorbing state that all arcs that go to the exit node transit into. The fundamental matrix of this absorbing Markov chain gives the traversal probabilities of the non-repeated arcs. For each repeated arc it is necessary to do a separate calculation where the arc transits to the absorbing state. A computer program has been written for this calculation; Figure 3 is the output for the directed graph of Figure 1.

| ARC TAIL | ARC HEAD | REPEAT | BRANCH PROB. | TRAVERSAL PROB. |
|---|---|---|---|---|
| 6 | 12 | R | 1.0000 | 0.3333 |
| 12 | 3 | R | 0.5000 | 0.2500 |
| 11 | 13 | | 1.0000 | 0.0833 |
| 13 | 16 | | 1.0000 | 0.1667 |
| 14 | 16 | | 1.0000 | 0.2500 |
| 7 | 11 | | 0.5000 | 0.0833 |
| 8 | 14 | | 1.0000 | 0.1667 |
| 1 | 2 | | 1.0000 | 1.0000 |
| 2 | 3 | | 0.5000 | 0.5000 |
| 12 | 16 | | 0.5000 | 0.5000 |
| 2 | 4 | | 0.5000 | 0.5000 |
| 4 | 7 | | 0.3333 | 0.1667 |
| 4 | 8 | | 0.3333 | 0.1667 |
| 4 | 9 | | 0.3333 | 0.1667 |
| 7 | 10 | | 0.5000 | 0.0833 |
| 9 | 14 | | 0.5000 | 0.0833 |
| 9 | 15 | | 0.5000 | 0.0833 |
| 10 | 13 | | 1.0000 | 0.0833 |
| 15 | 16 | | 1.0000 | 0.0833 |
| 3 | 5 | R | 0.5000 | 0.3333 |
| 3 | 6 | R | 0.5000 | 0.3333 |
| 5 | 12 | R | 1.0000 | 0.3333 |

Figure 3 - Traversal probabilities for Figure 1.

Given the traversal probability, $p_j$, and the expected number of errors, $\mu_j$, for each arc, the expected number of errors detected (and corrected) can be calculated. The expected number of errors detected by the first input is $\Sigma\, p_j\mu_j$ where the summation is over all the arcs of the graph. After the first input the expected number of errors in arc $j$ is $\mu_j(1-p_j)$. The expected number of errors detected by the second input is $\Sigma\, \mu_j p_j(1-p_j)$. In general, for the $k\underline{th}$ input the expected number of errors detected is $\Sigma\, \mu_j p_j(1-p_j)^{k-1}$.

If the number of errors on the arcs are mutually independent and the variances are finite and known, it is possible to calculate the standard deviation of the number of errors detected by the first input. For subsequent inputs, the standard deviation can not be easily calculated because the number of errors on the arcs are no longer independent.

## SIMULATION

The error detection model has been implemented as a FORTRAN simulation model. The directed graph is input to the simulation as a node-arc incidence matrix. Lacking detailed information about the distributions of the pertinent variables in actual systems, we saw no reason to establish statistical dependencies among the variables. Thus, the random variables were chosen to be independent and to possess the Markov property. This also makes the model more tractable for obtaining an analytical solution.

The number of instructions per arc is an exponential random variable truncated to an integer. Errors are inserted by making the number of instructions between errors an independent exponential random variable; this gives a Poisson distribution of errors per interval of instructions. Errors are inserted by scanning the arcs of the node-arc incidence matrix by columns until the count of instructions from the last error equals the random number.

An input is a sequence of random numbers that determine which arc to traverse at each branch node. For each branch node the probability of taking each arc is equal. This could be changed to test the sensitivity of error detection to different branch probabilities. With directed cycles there is a possibility of paths of unbounded length; this is prevented by excluding inputs that traverse more than a certain large number of arcs.

The correction times for errors are exponential. If many programmers work on error correction with each correcting only a small number of errors, the effect of experience on error correction may be small so that a constant correction rate corresponding to the exponential would be appropriate. If few programmers work on corrections, experience would be a factor and an increasing correction rate distribution would be appropriate. For example,

the log-normal is sometimes used to represent the distribution of hardware repair times (4).

The execution times of instructions are exponential. It was assumed that the execution time of an instruction does not depend on past instruction times. This assumption may not hold if the programmer tends to sequence his instructions in certain patterns.

When an error is corrected, it is determined randomly if a new error is to be introduced. If a new error is to be introduced it is inserted in a randomly chosen arc with each arc having equal probability of selection.

The simulation is written so that any distribution can be changed for the purpose of sensitivity analysis. The choice of distributions may have significant effect on the simulation results for a given structure; however, since the objective is to evaluate results on a relative basis across various structures, the choice of distributions does not seem to be critical.

For each input, data are collected on the number of errors detected, number of new errors, number of remaining errors, number of arcs traversed, number of arcs traversed for the first time, time to execute instructions and time to correct errors.

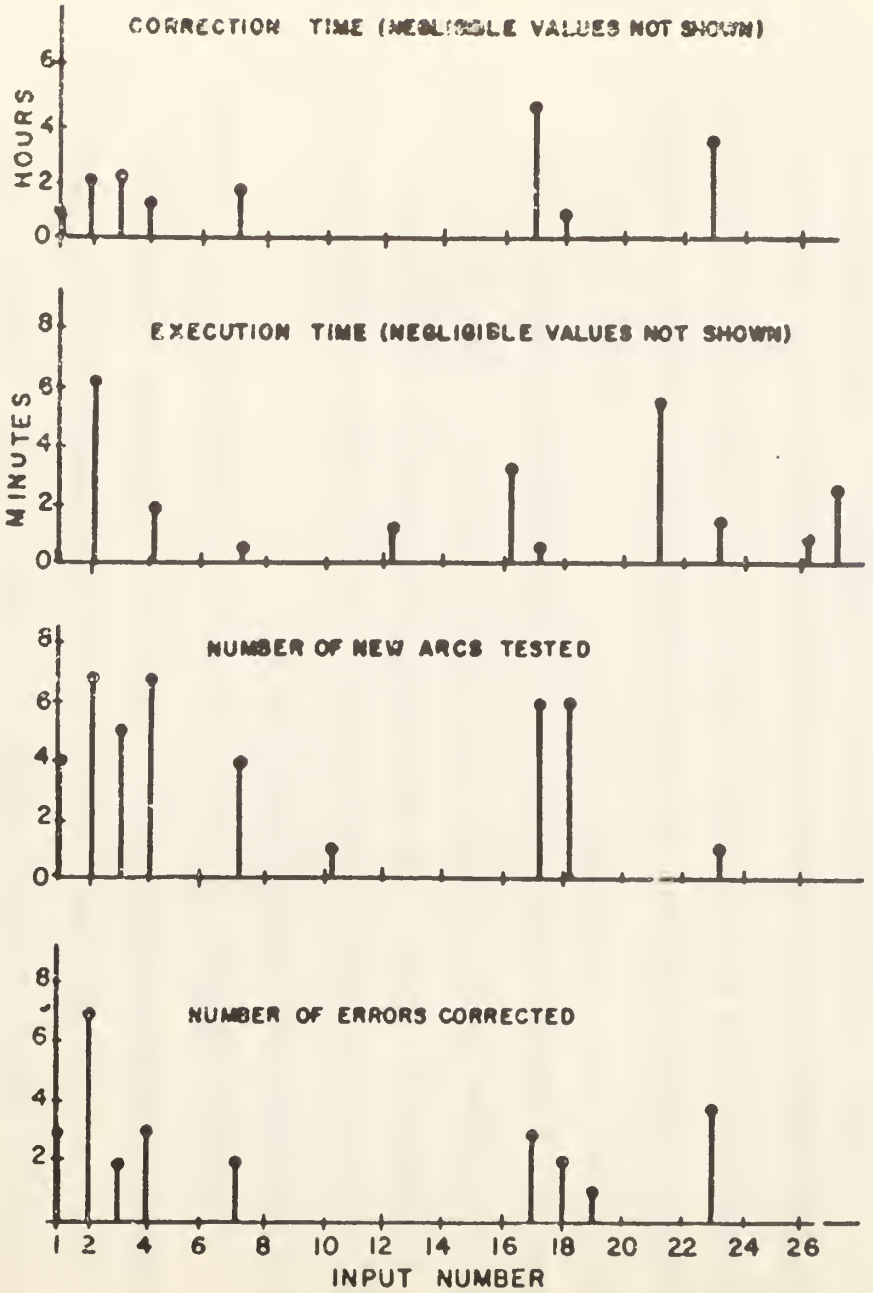
The simulation program was written so that it would be possible to generate random times for each instruction executed and for each error corrected as the simulation proceeds. However, if the instruction times and correction times are independent and identically distributed (e.g., as described above) then it is possible and computationally desirable to count the number of instructions executed and the number of errors corrected and then use the Law of Large Numbers to get a very good approximation of the distribution of each total time.

## ANALYSIS OF SELECTED SIMULATION RESULTS

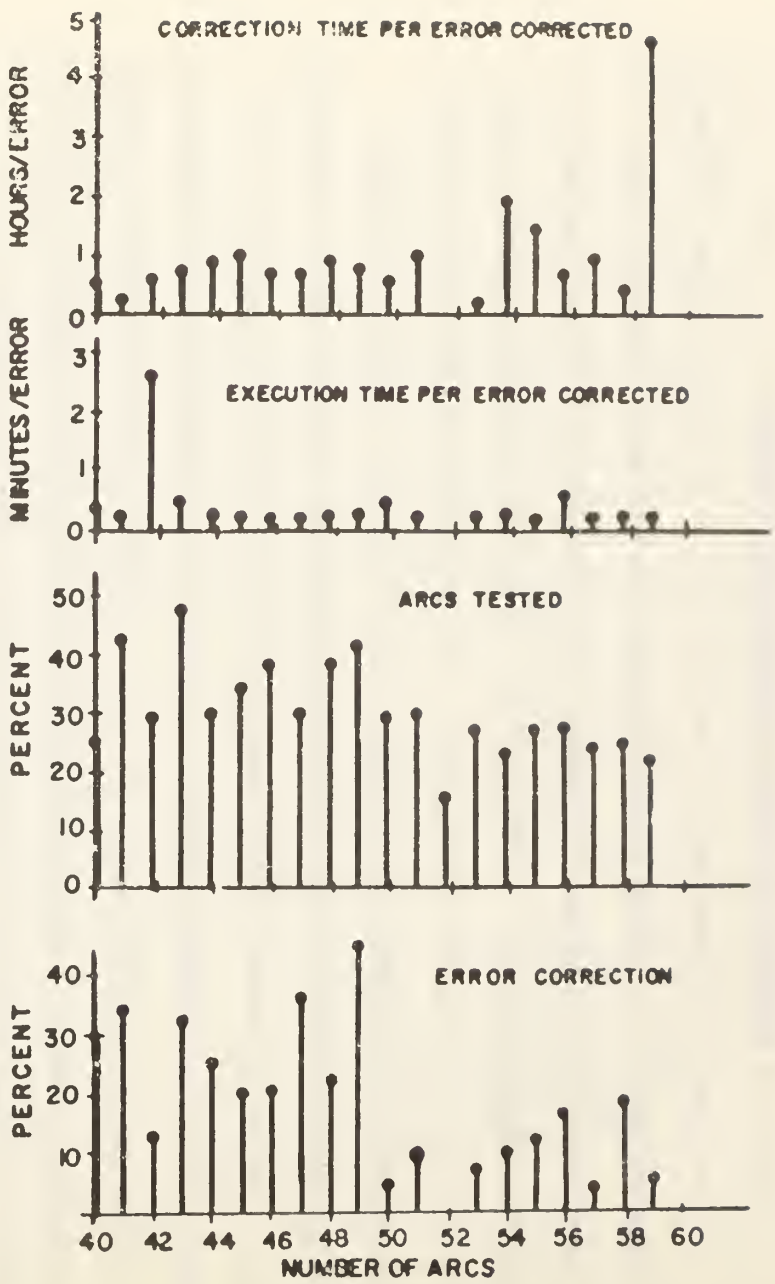
The simulation model was run on the Naval Postgraduate School IBM 360/67. Typical output from 1) submitting a sequence of inputs to a single program, and 2) submitting three inputs to a sequence of increasingly complex programs is described and discussed. The former case is illustrated by a program, with 30 nodes, 40 arcs and 435 instructions. Errors were initially placed on 30 instructions. A sequence of 30 inputs (i.e., 30 paths through the program determined by random numbers) were submitted to the program. Recall that each input detects all errors on its path; each error is corrected but there is some probability (in this example it was small) that new errors are introduced by correcting detected errors. For each input the number of errors corrected, the number of new arcs tested, execution time (related to computer costs) and correction time (related to personnel costs) are shown in Figure 4.

Figure 4 illustrates the action of the model. Initially there are many errors detected with each input; then the number of errors detected decreases as errors are corrected. Note that many paths do not traverse any new arcs. Although the paths through the program are, in general, different from previous paths, these paths may involve only arcs that have been previously traversed. In the testing of actual software, after an initial burst of errors there are often long periods with no error detection followed by a new group of errors. The new group of errors results from testing previously untested parts of the program. The error detection model captures this phenomenon, for example see Figure 4.

Figure 5 shows the results of running three successive inputs into 20 programs of increasing complexity. The initial program had 30 nodes and 40 arcs; the complexity was increased by adding an additional arc to each successive problem. For each problem the number of instructions and the




Simulation Results Obtained by Varying Inputs and Keeping Complexity Constant




Simulation Results Obtained by Varying Complexity and Using Same Inputs

Figure 5

number and location of errors was different. Because the number of errors and arcs varied, the results in Figure 5 are normalized. One would expect that the number of errors detected would decrease as the number of arcs increased; however, since the number of errors changed it was not possible to draw firm conclusions from only 20 runs.

## REFERENCES


(1) Boehm, Barry W., "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, pp. 48-59.

(2) Boehm, Barry W., "The High Cost of Software," in Proceedings of a Symposium on the High Cost of Software, Jack Goldberg (ed), Stanford Research Institute, Menlo Park, California, 1973, pp. 27-40.

(3) Stucki, L. G., "Automatic Generation of Self-Metric Software," Record of 1973 IEEE Symposium on Computer Software Reliability, New York City, April 30-May 2, 1973, pp. 94-100.

(4) Von Alven, William H., (ed), Reliability Engineering, Arinc Research Corporation, Prentice-Hall, 1964, pp. 155-156.

INITIAL DISTRIBUTION LIST

| | No. of Copies |
|---|---|
| Dean of Research<br>Code 023<br>Naval Postgraduate School<br>Monterey, California 93940 | 2 |
| Defense Documentation Center<br>Cameron Station<br>Arlington, Virginia 22314 | 12 |
| Library (Code 0212)<br>Naval Postgraduate School<br>Monterey, California 93940 | 2 |
| Library (Code 55)<br>Naval Postgraduate School<br>Monterey, California 93940 | 2 |
| W. R. Church Computer Center<br>Ingersoll Hall<br>Naval Postgraduate School<br>Monterey, California 93940 | 1 |
| Computer Sciences Department<br>Naval Electronics Laboratory Center<br>271 Catalina Boulevard<br>San Diego, California 92152 | |
| Mr. A. E. Beutel | 1 |
| Mr. Joseph Dodds | 1 |
| Naval Electronics Laboratory Center Library | 1 |
| Naval Air Development Center<br>Warminster, Pennsylvania 18974 | |
| Mr. H. Stuebing | 2 |
| Mr. R. Pariseau | 2 |
| Mr. Marvin Denicoff<br>Office of Naval Research<br>Department of the Navy<br>Arlington, Virginia 22217 | 1 |
| Fleet Combat Direction Systems Support Activity<br>200 Catalina Boulevard<br>San Diego, California 92147 | |
| Mr. M. Griswold | 1 |

| | |
|---|---|
| Captain Stephan Ruth (SC)<br>6204 Colmac Drive<br>Falls Church, Virginia 22044 | 1 |
| Stanford Research Institute<br>Menlo Park, California 94025 | |
| Mr. D. B. Parker | 1 |
| Mr. R. E. Keirstead | 1 |
| Professor D. Barr | 1 |
| Professor D. Gaver | 1 |
| Professor G. Barksdale | 1 |
| Professor G. Brown | 1 |
| Professor J. Esary | 1 |
| Professor M. Kline | 1 |
| Professor V. M. Powers | 1 |
| Professor U. Kodres | 1 |
| Professor R. Richards | 1 |
| Professor D. Williams | 1 |
| Professor M. Thomas | 1 |
| Professor G. Bradley | 15 |
| Professor G. Howard | 15 |
| Professor N. Schneidewind | 15 |